Introduction to SQL

Class 17

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, REPEAT, LOOP), ,Stored functions ,Stored Procedures
  - Views
  - Indexes, Cursors, Triggers, Events
  - ACID
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases, CAP
  - How SQL databases internally works

Course Overview

- Data modelling and database design
  - What is Data Model – (Steps Involved, Conceptual Data Modelling, Database design)
  - Entity Relationship model
  - Elements of entity relationship model
  - Entities, Subtypes, supertypes
  - Regular, Strong, Weak entities
  - Identifying and modelling entities
  - Relationships – Modelling, defining and types of relationships
  - Minimum and maximum relationships
  - Attributes- Finding attributes, meaningful components for attributes
  - Diagrammatic conventions
  - Transforming entity relationship model into relational schema
  - Logical database design (Top-Down Approach-E/R Modelling, Bottom-Up Approach-Normalization)
  - Difference between top-down and bottom-up approach
  - Normalization
    - Need for normalization
    - Forms of normalization – 1nf,2nf,3nf, Boyce Code Normalization
  - Case studies and discussion
  - Designing few real life databases

Questions for ER Diagrams

Explain the conversion of ER diagrams to tables in DBMS (tutorialspoint.com)
ER Diagrams to Tables | Gate Vidyalay
ER Diagrams to Tables | Practice Problems | Gate Vidyalay
total participation in dbms with example | Gate Vidyalay
Weak Entity and Strong Entity in DBMS | PrepInsta

NORMALIZATION

Why we need normalization

- A large database defined as a single relation may result in data duplication. This repetition of data may result in:
  - Making relations very large.
  - It isn't easy to maintain and update data as it would involve searching many records in relation.
  - Wastage and poor utilization of disk space and resources.
  - The likelihood of errors and inconsistencies increases.
- So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties
- Normalization is a process of decomposing the relations into relations with fewer attributes.

What is normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

First Normal From

- A table is in the first normal form iff
  - The domain of each attribute contains only ***atomic values***, and
  - The value of each attribute contains only a ***single value*** from that domain.

In layman's terms. it means every column of your table should only contain *single values*

Example

- For a library

Patron ID	Borrowed books
C45	B33, B44, B55
C12	B56

1-NF Solution

Patron ID	Borrowed book
C45	B33
C45	B44
C45	B33
C12	B56

Example

- For an airline

Flight	Weekdays
UA59	Mo We Fr
UA73	Mo Tu We Th Fr

1NF Solution

Flight	Weekday
UA59	Mo
UA59	We
UA59	Fr
UA73	Mo
UA73	We
...	...

Implication for the ER model

- Watch for entities that can have multiple values for the same attribute
  - Phone numbers, …
- What about course schedules?
  - MW 5:30-7:00pm
    - Can treat them as *atomic time slots*

Second Normal Form

- A table is in 2NF iff
  - It is in 1NF and
  - no non-prime attribute is dependent on any proper subset of any candidate key of the table
- A ***non-prime attribute*** of a table is an attribute that is not a part of any candidate key of the table
- A ***candidate key*** is a minimal superkey

Example

- Library allows patrons to request books that are currently out

BookNo	Patron	PhoneNo
B3	J. Fisher	555-1234
B2	J. Fisher	555-1234
B2	M. Amer	555-4321

Example

- Candidate key is {BookNo, Patron}
- We have
  - Patron → PhoneNo
- Table is not 2NF
  - Potential for
    - Insertion anomalies
    - Update anomalies
    - Deletion anomalies

2NF Solution

- Put telephone number in separate Patron table

BookNo	Patron
B3	J. Fisher
B2	J. Fisher
B2	M. Amer

Patron	PhoneNo
J. Fisher	555-1234
M. Amer	555-4321

Third Normal Form

- A table is in 3NF iff
  - it is in 2NF and
  - all its attributes are determined only by its candidate keys and not by any non-prime attributes

Example

- Table BorrowedBooks

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

- Candidate key is BookNo
- Patron $\rightarrow$ Address

3NF Solution

- Put address in separate Patron table

BookNo	Patron	Due
B1	J. Fisher	3/2/15
B2	L. Perez	2/28/15

Patron	Address
J. Fisher	101 Main Street
L. Perez	202 Market Street

Another example

- Tournament winners

Tournament	Year	Winner	DOB
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 Sept. 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975

- Candidate key is {Tournament, Year}
- Winner →DOB

Boyce-Codd Normal Form

- Stricter form of 3NF
- A table *T* is in BCNF iff
  - for every one of its non-trivial dependencies $X \rightarrow Y$, *X* is a super key for *T*

- Most tables that are in 3NF also are in BCNF

Example

Manager	Project	Branch
Alice	Alpha	Austin
Alice	Delta	Austin
Carol	Alpha	Houston
Dean	Delta	Houston

- We can assume
  - Manager → Branch
  - {Project, Branch} → Manager

Example

Manager	Project	Branch
Alice	Alpha	Austin
Bob	Delta	Houston
Carol	Alpha	Houston
Alice	Delta	Austin

- Not in BCNF because Manager → Branch and Manager is not a superkey
- Will decomposition work?

A decomposition (I)

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston

- Two-table solution does not preserve the dependency {Project, Branch} → Manager

A decomposition (II)

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta
Dean	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston
Dean	Houston

- Cannot have two or more managers managing the same project at the same branch

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

An Example

Normalisation Example

- We have a table representing orders in an online store
- Each row represents an item on a particular order
- Primary key is {Order, Product}

- Columns
  - Order
  - Product
  - Quantity
  - UnitPrice
  - Customer
  - Address

Functional Dependencies

- Each order is for a single customer:
  - Order $\rightarrow$ Customer
- Each customer has a single address
  - Customer $\rightarrow$ Address
- Each product has a single price
  - Product $\rightarrow$ UnitPrice
- As Order $\rightarrow$ Customer and Customer $\rightarrow$ Address
  - Order $\rightarrow$ Address

2NF Solution (I)

- ***First decomposition***
  - First table

Order	Product	Quantity	UnitPrice

  - Second table

Order	Customer	Address

2NF Solution (II)

- ***Second decomposition***
  - First table

Order	Product	Quantity

  - Second table

Order	Customer	Address

  - Third table

Product	UnitPrice

3NF

- In second table

Order	Customer	Address

  - Customer $\rightarrow$ Address
- Split second table into

Order	Customer

Customer	Address

Normalisation to 2NF

- Second normal form means no partial dependencies on candidate keys
  - {Order} $\rightarrow$ {Customer, Address}
  - {Product} $\rightarrow$ {UnitPrice}

- To remove the first FD we project over

  {Order, Customer, Address} (R1)

and

{Order, Product, Quantity, UnitPrice} (R2)

Normalisation to 2NF

- R1 is now in 2NF, but there is still a partial FD in R2

  $\{Product\} \rightarrow \{UnitPrice\}$

- To remove this we project over

{Product, UnitPrice} (R3)

and

{Order, Product, Quantity} (R4)

Normalisation to 3NF

- R has now been split into 3 relations - R1, R3, and R4
  - R3 and R4 are in 3NF
  - R1 has a transitive FD on its key

- To remove

  {Order} $\rightarrow$ {Customer} $\rightarrow$ {Address}

- we project R1 over
  - {Order, Customer}
  - {Customer, Address}

Normalisation

- 1NF:
  - {Order, Product, Customer, Address, Quantity, UnitPrice}
- 2NF:
  - {Order, Customer, Address}, {Product, UnitPrice}, and {Order, Product, Quantity}
- 3NF:
  - {Product, UnitPrice}, {Order, Product, Quantity}, {Order, Customer}, and {Customer, Address}

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Top-Down Database design vs Bottom-Up Database design

- Top-Down Design: In top-down design, the database design starts with a high-level conceptual model and progresses to lower levels of detail, such as logical and physical data models. This approach is best suited for complex database systems that require a comprehensive understanding of the data requirements before the database is built.
- Bottom-Up Design: In bottom-up design, the database design starts with existing database structures and data and then progresses to higher levels of abstraction, such as logical and conceptual data models. This approach is best suited for organizations that have existing databases and need to integrate or improve them, rather than starting from scratch.

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)